(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) **Offenlegungsschrift**

(10) **DE 100 32 962 A 1**

(51) Int. Cl.[7]:
G 06 F 17/30
G 06 F 15/173

(21) Aktenzeichen: 100 32 962.4
(22) Anmeldetag: 6. 7. 2000
(43) Offenlegungstag: 15. 3. 2001

DE 100 32 962 A 1

(30) Unionspriorität:

142489 06. 07. 1999 US
565979 04. 05. 2000 US

(71) Anmelder:
Matsushita Electric Industrial Co. Ltd., Osaka, JP

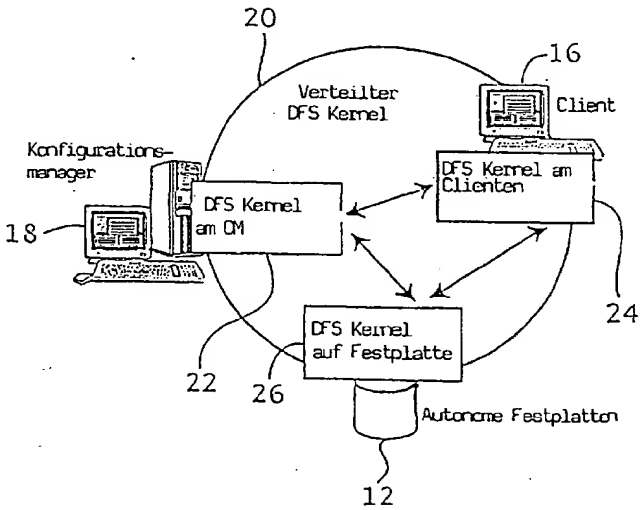
(74) Vertreter:
Hauck & Wehnert, 80336 München

(72) Erfinder:
Mukherjee, Sarit, Mount Laurel, N.J., US; Aref, Walid G., New Brunswick, N.J., US; Kamel, Ibrahim M., Monmouth Junction, N.J., US; Braun, David A., Dehville, N.J., US

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

(54) Echtzeitfähiges verteiltes Dateisystem

(57) Es wird ein verteiltes Dateisystem (10) mit autonomen Platten (12) beschrieben. Das Dateisystem (10) umfaßt autonome Platten (12), die Applikationsdaten speichern. Legacyattributplatten speichern Metadaten des Dateisystems (10) und die Verzeichnisstruktur. Die Verzeichnisstruktur wird auf den Legacyattributplatten im ursprünglichen Dateisystem abgelegt. Einer der Klienten (16) des Dateisystems wird als Konfigurationsmanager (18) zur Zugriffsüberwachung für das System (10) verwendet. Das verteilte Dateisystem verwendet Agenten (22, 24, 26) um Echtzeitanwendungen und konkurrierende Lese/Schreib-Zugriffe auf Dateien zu unterstützen.



DE 100 32 962 A 1

Beschreibung

Die Erfindung betrifft allgemein verteilte Dateisysteme und insbesondere die Architektur und Verwirklichung echtzeitfähiger verteilter Dateisysteme.

Fortschritte bei der Netzwerk- und Speichertechnologie haben zusammen mit der Digitalisierung von Multimediaanwendungen schnelle und große Server notwendig gemacht. Diese Server werden üblicherweise als an ein Netzwerk angeschlossene Speicher verwendet. Verschiedene Klienten (client) hosts können online über das Netzwerk darauf zugreifen. Die Klienten binden das Dateisystem ihrer hosts ein und benutzen die Funktionalität der Server durchgehend.

Es gibt drei Arten von Multimediaservern: Zentralisierte, verteilte oder serverlose. Bei einem zentralisierten Server steuert ein einziger zugewiesener Knoten den Zugriffsvorgang und alle weiteren Funktionen bezüglich Dateioperationen und Datensicherheit. Bei einer verteilten Serverumgebung teilt sich eine Gruppe bestimmter Knoten die Kapazitäten und Funktionen des Servers. Bei einem serverlosen System sind alle Klienten und Speichereinrichtungen direkt an das Netzwerk angeschlossen.

Im allgemeinen umfaßt ein in einer Serverumgebung verwirklichtes verteiltes Dateisystem eine verteilte Verzeichnisstruktur, die unabhängig von dem den einzelnen Rechnern zugeordneten Dateisystem ist. Die verteilte Verzeichnisstruktur ist auf den einzelnen Rechnern als Kopie abgelegt. Der mit dem Kopieren und Ablegen der verteilten Verzeichnisstruktur verbundene Verwaltungsaufwand ist sehr groß und mindert die Leistungsfähigkeit des gesamten Dateisystems.

Darüber hinaus mangelt es bekannten verteilten Dateisystemen an einem Verfahren zum Steuern des Zugriffs auf die Bandbreite. Steigern die Klienten die Zahl an Zugriffen auf das Dateisystem, steigt die Inanspruchnahme der Systemresourcen des Dateisystems, weshalb es unmöglich ist, Echtzeitanwendungen zu unterstützen.

Man benötigt deshalb ein verbessertes Verfahren zur Verwirklichung eines verteilten Dateisystems. Bei diesem System soll der mit dem Speichern der verteilten Dateisystemverzeichnisstruktur und dem Speichern von Anwendungsdaten verbundene Verwaltungsaufwand reduziert werden. Darüber hinaus soll die Leistungsfähigkeit des verteilten Dateisystems gesteigert und eine Skalierbarkeit des Speichersystems erreicht werden. Weiter soll dieses System netzwerk- und protokollunabhängig sein. Schließlich benötigt man ein verteiltes Dateisystem, das echtzeitfähig ist.

Diese Aufgabe wird durch die in den Ansprüchen gekennzeichnete Erfindung gelöst. Vorteilhafte Weiterbildungen sind Gegenstand der Unteransprüche.

Die vorliegende Erfindung schafft ein verteiltes Dateisystem zum Speichern und Abrufen von Informationen auf bzw. von einem oder mehreren Speichersystemen über ein Netzwerk von einem oder mehreren Host-Systemen aus. Ein bevorzugtes Speichersystem ist eine als autonome Festplatte (AD) bezeichnete Vorrichtung. Die AD ist eine Festplatte oder ein anderes Speichermedium, das einen entsprechenden Rechner aufweist. Da das Dateisystem vorteilhafterweise nur geringe Verarbeitungsanforderungen an diesen Rechner stellt, kann man die AD mit relativ kleinen, kostengünstigen Prozessoren verwirklichen.

Das erfindungsgemäße Dateisystem hat einen Speichersystemkernel oder -agenten, der im AD Speichersystem sitzt. Der Speichersystemkernel umfaßt ein freies Listenmanagementsystem, das die physikalische Lage des Speichers von auf dem Speichersystem abgelegten Informationen ermittelt. Das Dateisystem arbeitet mit einem Verzeichnisstruktursystem, das am Hostsystem abgelegt ist und eine logische Organisation mehrerer Dateien festlegt, die den am Speichersystem abgelegten Informationen entsprechen. Das Dateisystem kann man mittels existierender dem Host zugeordneter Dateisysteme verwirklichen.

Ein Legacyattributdatenspeicher ist mit dem Netzwerk verbunden und speichert Metadaten für die auf dem Speichersystem abgelegte Information. Die Hostsysteme können auf diese Metadaten zugreifen, um die physikalische Lage des die auf den AD gespeicherten Informationen enthaltenden Speicher zu ermitteln.

Weiter weist das Dateisystem einen Klientenkernel oder -agenten auf, der im Hostsystem liegt und Zugriff auf die Metadaten des Datenspeichers mit den Legacyattributen hat. Um jeweilige Dateien den entsprechenden physikalischen Speicherstellen zuzuordnen, wirkt der Klientenkernel mit dem Verzeichnisstruktursystem zusammen. Mittels dieser Informationen kann ein Host Information vom Speichersystem abrufen und über das Netzwerk erhalten.

Die in einer gegenwärtig bevorzugten Ausführungsform der Erfindung verwendeten autonomen Festplatten (AD) schaffen Flexibilität bei der Auslegung eines Datei-Servers. Man kann mit ihnen ein verteiltes Dateisystem aufbauen, indem Aufgaben auf mehrere AD delegiert werden. Ein serverloses Dateisystem kann durch Ausführen von Dateisystemoperationen in den AD verwirklicht werden. In die AD kann ein Sicherheitsmodul eingebaut werden, um einen unberechtigten Zugriff auf das System zu verhindern. Zur Verwirklichung eines AD kann man verschiedene Hard- und Softwaremittel einsetzen.

Die in dieser Erfindung beschriebene Architektur des verteilten Dateisystems (DFS) baut auf die AD als Baustein auf. Das DFS hat eine verteilte Architektur mit mehreren über ein Netzwerk verbundenen Speichergeräten. Die Userhosts sind ebenfalls an dieses Netzwerk angeschlossen. Ein als Konfigurationsmanager bezeichneter Userhost ist so ausgestattet, daß er die verteilten, DFS-spezifischen Datenstrukturen und Systemkonfigurationen enthält und die Zugriffssteuerung durchführt. Der Kernel des DFS ist auf die autonomen Festplatten, die Userhosts und den Konfigurationsmanager verteilt. Der Kernel sorgt dafür, daß die grundlegenden Operationen des Systems für den Benutzer transparent sind.

Die AD ist eine Festplatte oder ein anderes Speichermedium mit einem kleinen programmierbaren Speicher und kann durch aktive, mit dem Netzwerk verbundene Platten, normale Workstations oder andere Mittel verwirklicht werden. Die AD führt einige einfache Funktionen des Dateisystems aus. Diese Funktionen werden als Teil des DFS Kernels ausgeführt, der auf der Platte läuft. Darüber hinaus hat die AD eine Netzwerkschnittstelle, über die sie direkt mit dem Netzwerk verbunden werden kann.

Die DFS-Daten werden vorzugsweise in Volumes organisiert. Jedes Volume besteht aus einer oder mehreren autonomen Datenfestplatten, die ein Typ von autonomen Festplatten sind. Eine Datei wird auf die Datenfestplatten des Volumes verteilt. Die Metadaten für dieses Volume des Filesystems werden auf einer anderen autonomen Festplatte gespeichert, die als Legacyattributplatte (LAD) bezeichnet wird. Die verteilte Verzeichnisstruktur des Verteilsystems wird auf der LAD mit ihrem normalen Dateisystem gespeichert. Dadurch kann das DFS Steuervorgänge und Daten getrennt behandeln, wodurch der Verwaltungsaufwand reduziert ist. Das Dateisystem unterstützt Echtzeitanwendungen und ermöglicht skalierbare Datenspeicher.

Das oben erwähnte System ist nur beispielhaft. Erfindungsgemäße Systeme können auf vielfältige Weise verwirklicht werden.

Dies wird aus der nachfolgenden Zeichnungsbeschrei-

bung ersichtlich. In der Zeichnung zeigen:

Fig. 1(a–b) die Architektur einer gegenwärtig bevorzugten Ausführungsform des verteilten Dateisystems,

Fig. 1(c) eine Verwirklichung autonomer Festplatten mit einem PC,

Fig. 2 ein Volume eines verteilten Dateisystems,

Fig. 3 einen Prozeß zum Einloggen in ein verteiltes Dateisystem,

Fig. 4 einen Lesevorgang beim verteilten Dateisystem,

Fig. 5 einen Schreibvorgang beim verteilten Dateisystem,

Fig. 6 verschiedene Volume-Konfigurierungen einer gegenwärtig bevorzugten Ausführungsform des verteilten Dateisystems,

Fig. 7 eine Verzeichnisstruktur einer gegenwärtig bevorzugten Ausführungsform des verteilten Dateisystems,

Fig. 8 eine detailliertere Darstellung eines Lesevorgangs bei einem DFS und

Fig. 9 eine detaillierterer Darstellung eines Schreibvorgangs bei einem DFS.

Fig. 1(a) zeigt ein Dateisystem **10**, das eine verteilte Architektur hat. Bei dieser Architektur sind mehrere Speichergeräte vorgesehen, die als autonome Festplatten (AD) **12** bezeichnet und an ein Netzwerk **14** angebunden sind. In einer bevorzugten Ausführungsform ist das Netzwerk **14** ein Hochgeschwindigkeitsnetzwerk, das in der Lage ist, eine große Bandbreite für das System bereitzustellen. Natürlich kann man das System unabhängig von Art und Geschwindigkeit des Netzwerks verwirklichen. Userhosts **16** sind an das Netzwerk **14** angeschlossen. Ein Konfigurationsmanager (cm) **18** ist ebenfalls an das Netzwerk angeschlossen. Er hält und verteilt systemspezifische Datenstrukturen und Konfiguration. Der cm kann selbst ein Userhost sein.

Fig. 1(b) zeigt den verteilten DFS-Kernel **20**. Der Kernel **20** ist auf cm 18, den Userhost **16** und AD **12** verteilt. Die einzelnen Teile des DFS-Kernels auf dem cm 22, dem User host 24 und den AD **26** arbeiten nahtlos zusammen, so daß die User hosts 16 das zugrundeliegende, verteilte Dateisystem transparent sehen. Die User hosts merken nichts von den Protokollen und Verfahren, die zum Lesen und Schreiben von Daten auf das DFS verwendet werden.

Die autonome Festplatte (AD) **12** ist eine aktive Platte. Vorzugsweise enthält sie einen kleinen programmierbaren Prozessor und einen Speicher. Die AD **12** kann durch aktive, aus Netzwerk angebundene Festplatten, normale Workstations oder andere Mittel verwirklicht werden. Sie hat eine Netzwerkschnittstelle zur direkten Anbindung an das Netzwerk **14**. Der Prozessor der AD **12** führt einige Funktionen des Dateisystems durch. Diese Funktionen werden als der auf der Platte **26** laufende Teils des DFS-Kernels ausgeführt. Sie umfassen vorzugsweise die Verwaltung der Freiplätze, Netzwerkprotokollverarbeitungen und Paketübertragungen, Plattenanforderungsteuerungen und Zugriffsüberwachungen.

Fig. 1(c) zeigt eine Verwirklichung mehrerer AD **12** in Form eines PC mit einem einzigen Prozessor **25**, einem Speicher **27** und einer Netzwerkschnittstellenkarte (NIC) **28** sowie einem Ein-/Ausgabebus **30**. Der Prozessor **25** führt die für jede AD **12** nötigen Dateisystem- und Verarbeitungsfunktionen durch. Der Speicher **27** bzw. die NIC **28** stellen den Speicherplatz für Berechnungen bzw. die Verbindung zu einem Netzwerk **14** bereit. Der Prozessor **25**, der Speicher **27**, die NIC **28** und die Platten **12** sind über einen gemeinsamen Ein-/Ausgabebus **30** verbunden. Auf einem PC können mehrere AD **12** verwirklicht werden, solange die gesamte Festplattenbandbreite nicht insgesamt die Netzwerkbandbreite überschreitet, die von PC und NIC gemeinsam bereitgestellt wird. Die Adresse einer AD **12** ist einfach das tuple aus der host Adresse des PC und der Identifikationsnummer (ID) der Platte.

Anders als bei einem Legacydateisystem, bei dem der Rechner die Liste freier Blöcke verwaltet und auf Anforderungen Blöcke in der Liste belegt oder freigibt, werden diese Funktionen nun vom auf der AD **12** liegenden Teil des DFS-Kernel durchgeführt. Weiter übernimmt die AD **12** einen Teil der Protokollverarbeitung.

In einer bevorzugten Ausführungsform ist das Netzwerkprotokoll das Internetprotokoll; es sind aber auch andere Protokolle möglich. Die Datenverbindung für die AD **12** kann auf Fibre Channel oder einer anderen MAC (z. B. Fast or Gigabit Ehernet) aufbauen. Der in der Platte vorgesehene Prozessor führt eine intelligente Anforderungsverwaltung durch. Der Verwaltungsalgorithmus sollte programmierbar sein; natürlich kann man auch einen nicht programmierbaren Algorithmus verwenden. Da die AD **12** direkt an das Netzwerk **14** angeschlossen ist, kann sie Hackerangriffen ausgesetzt sein. Sie sollte deshalb eigene Zugriffsrechtüberwachungen durchführen.

Fig. 2 zeigt eine vereinfachte Darstellung der Volumeunterteilung **46** eines DFS. Das Dateisystem speichert die Applikationsdaten und die Metadaten getrennt. Es werden zwei Arten von Platten verwendet: Autonome Datenplatten (ADD) **42** und Legacyattributdatenplatten (LAD) **44**. Ein Volume **46** eines DFS besteht aus mehreren ADD **42** und mindestens einer LAD **44**. Die Dateien auf einem Volume **46** werden über mehrere ADD **42** verteilt. Mehrere Volumes können sich eine LAD **44** teilen, indem die LAD **44** in mehrere Partitionen unterteilt wird, eine für jedes DFS Volume.

Der Konfigurationsmanager (cm) **18** eines DFS hält alle Metadaten des Dateisystem vorrätig, die die Volumekonfiguration und die Information über die Zugriffsrechte der Benutzer betreffen. Um in einem DFS auf ein Volume **46** zugreifen zu können, loggt sich der Benutzer durch den cm 18 in das System ein. Nach dem Einloggen in das DFS durch den cm 18 erhält der Benutzer **16** Zugriff auf diejenigen Volumes **46**, für die er authorisiert wurde. Der cm 18 informiert weiter die LAD **44** und die ADD **42** über die aktiven Benutzer, die auf sie zugreifen dürfen.

Fig. 3 zeigt den Einloggvorgang in das DFS. Möchte ein neuer Benutzer **16** auf ein DFS Volume zugreifen, loggt er sich zuerst in das System ein, indem er dem Konfigurationsmanager **18** eine login-Anforderung **50** sendet. Der cm 18 überprüft den Benutzer **16** und erlaubt ihm, diejenigen Volumes **46** zu benutzen, auf die er (schreibend oder lesend) Zugriff haben darf. Der cm 18 führt den Authorisierungsvorgang nur auf der Ebene der Volumes durch.

Hat der cm 18 den Benutzer **16** authorisiert, setzt er die ADD **42** von der Identität des Benutzers in Kenntnis; dies ist bei 52 dargestellt. Die ADD **42** fügen die Identität des Benutzers auf ihrer entsprechenden Zugriffssteuerungsliste hinzu. Kommt eine Anforderung zu einer ADD **42**, überprüft sie zuerst, ob diese Anforderung mit einer entsprechend gültigen Identität versehen ist, und verarbeitet erst dann die (Lese- oder Schreib-)Anforderung. Ist die Identität des Benutzers nicht in der Zugriffssteuerliste der ADD vorhanden, wird die Anforderung **50** ohne Nachricht an den absendenden Benutzer **16** abgewiesen.

Fig. 4 zeigt auf höherer Ebene einen Lesevorgang für ein DFS. Der Klient **16** darf diesen Vorgang nur durchführen, nachdem er eingeloggt ist und zum Zugriff auf das Volume authorisiert wurde, wie oben bereits erläutert. Bei einer Leseanforderung einer Applikation kontaktiert der auf den Benutzerhost **16** laufende DFS-Kernel zuerst die LAD **44** des Volumes **46** hinsichtlich der Attribute der zu lesenden Datei; dies ist bei 60 dargestellt. Mittels der Dateiindextabelle (i-node) übersetzt der DFS-Kernel am Benutzer host 16 eine Leseanforderung in eine Gruppe einer oder mehrerer Sende-

anforderungen für eine oder mehrere ADD **42**. Die Sendeanforderungen umfassen die Adresse der ADD **42** und die davon zu lesenden Blöcke. Die Anforderung wird dann an die ADD **42** gesendet; dies ist bei 64 dargestellt. Die ADD **42** senden die geforderten Datenblöcke zurück; dies ist bei 66 dargestellt.

Fig. 5 zeigt auf höherer Ebene einen Schreibvorgang **70** bei einem DFS. Der Schreibvorgang **70** unterscheidet sich vom Lesevorgang **58**, da eine Dateiindextabelle für die geforderte Datei auf der LAD **44** für das Volume **46** (das der Benutzer **16** schreiben möchte), nicht existieren muß. Der Benutzer **16** sendet eine Anforderung, eine Datei zu erzeugen; dies ist bei 72 dargestellt. Daraufhin sendet die LAD **44** die Adresse eines freien Blocks an den Benutzer **16**; dies ist bei 74 dargestellt. Der Benutzer **16** erzeugt eine Dateiindextabelle **75**, mit der die Datei geschrieben wird und sendet die Tabelle an die LAD **44** des Volumes; dies ist bei 76 dargestellt. Für jeden Schreibvorgang den der am Benutzer **16** laufende DFS-Kerne durchführen möchte, wird ein Datenblock an die ADD **42** geschickt; dies ist bei 78 dargestellt. Die ADD **42** wählt für die Daten einen freien Block aus der entsprechenden Liste der freien Blöcke und sendet die Blockadresse an den Benutzer **16**; dies ist bei 80 dargestellt. Der Benutzer erzeugt die Dateiindextabelle (i-node) **75** aus diesen Informationen und schickt sie an die LAD **44** des Volumes; dies ist in Schritt **76** dargestellt.

Das DFS verwendet Datenaufteilung, so daß Applikationsdaten und Metadaten des Dateisystems getrennt gespeichert werden. Die ADD **42** speichern die Applikationsdaten, und die LAD **44** speichert die Metadaten. Die für eine Applikation bei einem Volume **46** zur Verfügung stehende Bandbreite wird im wesentlichen von der ADD **42** und nicht von den LAD **44** begrenzt. Erhöht man die Verteilungsgröße eines Volumes, so steigt die gesamte Bandbreite dieses Volumes an.

Fig. 6 zeigt verschiedene Volumeausbildungen, die im DFS-Verwendung finden können. Ein DFS Volume **46** kann aus homogenen Platten bestehen, wie es bei 90 dargestellt ist. Das Volume kann heterogene Platten haben, wie es bei 92, 94 und 98 dargestellt ist. Dadurch kann man das Dateisystem an Veränderungen der Plattentechnologie anpassen. Die langsamste ADD **42** eines Volumes begrenzt die Gesamtbandbreite dieses Volumes; die Gesamtbandbreite des Volumes entspricht dem Produkt aus der Anzahl an Platten mit der niedrigsten Bandbreite dieser Platten.

Mit DFS kann man Überschneidungen von Volumes zulassen. Die Volumes **49** und **98** überschneiden sich beispielsweise in der Platte **96**. Eine ADD **42** kann Teil mehrerer Volumes sein. Solche ADD werden als geteilte ADD (SADD) **96** bezeichnet. Eine SADD **96** ist nicht partioniert und bedient gleichzeitig alle Volumes, zu denen sie gehört. Dadurch, daß Überschneidungen möglich sind, kann DFS diejenigen Platten effizienter nutzen, die wesentlich schneller sind als andere Platten des Systems. Die überschneidenden Volumes **94** und **96** sind als inhomogen dargestellt, natürlich kann man auch homogene Volumes überschneiden.

Die LAD **44** hat im DFS zwei Aufgaben. Zum einen ist sie als ursprüngliches Dateisystem des host Betriebssystems (z. B. NTFS bei Windows NT, ext2fs bei Linux) konfiguriert, um die gesamte Verzeichnisstruktur und die zugehörigen Funktionen zu gewährleisten. Dies bedeutet, daß der Zugriff eine Sicherheitsstufe ist, nämlich in Form von Zugriffsrechten auf Verzeichnisse und Dateien. Als weiteres speichert die LAD **44** die Metadaten für das DFS. Jede DFS Datei hat einen entsprechenden Eintrag auf der LAD **44**.

Eine DFS Datei besteht aus einer Liste virtueller Blöcke, die auf einem DFS Volume **46** gespeichert sind. Die Ablage der Blöcke im Volume **46** ist reihum (mit zufällig gewählter Startplatte für jede Datei). Die Blockgröße ist voreingestellt und für das gesamte Volume **46** unveränderlich. Diese unveränderliche Größe hat mehrere Zwecke. Erstens vereinfacht sie die Verwirklichung des Dateisystems. Zweitens ermöglicht sie eine besser vorhersehbare Zugriffszeit gegenüber Systemen mit variabler Blockgröße. Drittens ermöglicht sie eine deterministischere Pufferbelegung. Viertens ermöglicht sie eine deterministischere Platten- und Netzwerkauslastung. Fünftens verringert sie die Fragmentierung; die interne Fragmentierung ist gering, und externe Fragmentierung gibt es nicht.

Jeder virtuelle Block einer Datei hat eine virtuelle Blockadresse (VBA). Aus der VBA bildet das DFS die entsprechende logische Bockadresse (LBA) im Volume **46**. Da ein DFS Volume **46** üblicherweise aus mehreren ADD **42** besteht (auf denen die Blöcke physikalisch abgelegt werden), ist jede VBA auf die LBA der entsprechenden ADD **42** im Volume **46** abgebildet. Deshalb benötigt man eine Abbildungstabelle, die VBA auf LBA und die Identifikationsnummer der ADD **42** abbildet.

Diese Abbildungstabelle wird als Datei auf der LAD **44** abgelegt (im ursprünglichen Dateisystem) und als Metadatum für den Dateizugriff auf DFS verwendet. Diese Datei wird als Attributdatei bezeichnet. Obwohl bislang erwähnt wurde, daß für jede VBA ein Eintrag vorhanden ist, kann man auch Bereichsabbildungen anstelle Einzeladreßabbildungen ablegen. Um auf eine DFS Datei zuzugreifen, muß ein Benutzer **16** zuerst die entsprechende Attributdatei erhalten. Für Fehlertoleranz bei vollständiger Datenspiegelung (RAID Ebene **1**) im Speicherbereich kann man mehrere VBA-auf-LBA Abbildungstabellen in der Attributdatei speichern.

Die Attributdatei enthält mehrere Schlüsselinformationen für eine DFS Datei. Die Attribute werden in einer erweiterbaren Struktur gespeichert. Die in der Attributdatei enthaltenen Attribute können umfassen: Dateigröße, Bandbreitenanforderung, magische Zahl, Medientyp, Redundanzgrad und Dateiindextabelle.

Die Dateigröße ist die Größe der DFS Datei. Sie wird ausreichend häufig geändert, so daß sie zu jeder Zeit die korrekte Dateigröße wiedergibt (beispielsweise wenn eine Datei geschrieben wird, ändert sich die Größe fortlaufend). Beim Schreiben wird jeder Datei eine Standardbandbreiteninanspruchnahme zugewiesen. Das ist die Standardbandbreite, die dieser Datei beim Öffnen zugrundegelegt wird. Der Benutzer **16** kann die Bandbreiteninanspruchnahme mit speziellen Steuerbefehlen ändern. Man kann eine Datei mit einer anderen als der Standardbandbreite öffnen. Die magische Zahl definiert die Attributdatei, die, obwohl sie wie eine normale Datei erscheint, ein Metadatum für das DFS ist. Sie sollte deshalb auch gesondert behandelt werden. Der Medientyp definiert (z. B. Audio, Video und Bild) und den Kompressionsmechanismus (z. B. MPEG-2, DVCPro). Der Redundanzgrad (LoR) definiert den Redundanzgrad des Speichers, der für die DFS Datei vorgesehen ist. Die Attributdatei speichert genau so viele Indextabellen, wie der Redundanzgrad vorgibt.

Der cm 18 speichert die speziellen Volumeinformationen. Für jedes Volume **46** sollen die folgenden Informationen vorgehalten werden: Volumename, die für die Datenverteilungsgruppe vorgesehenen ADD, die Reihenfolge in der Datenverteilungsgruppe, die Volumeerzeugungszeit, die Volumegröße, der freie Speicherplatz im Volume, die Bandbreite des Volumes, die freie Bandbreite des Volumes, die Blockgröße und die Standardbandbreite zum Lesen oder Schreiben von Dateien dieses Volumes.

Anders als bei einem Legacydateisystem, bei dem der

Rechner die Liste freier Blöcke verwaltet und auf Anforderungen Blöcke in der Liste belegt oder freigibt, werden diese Funktionen nun vom auf der AD **12** liegenden Teil des DFS-Kernel durchgeführt. Weiter übernimmt die AD **12** einen Teil der Protokollverarbeitung.

In einer bevorzugten Ausführungsform ist das Netzwerkprotokoll das Internetprotokoll; es sind aber auch andere Protokolle möglich. Die Datenverbindung für die AD **12** kann auf Fibre Channel oder einer anderen MAC (z. B. Fast or Gigabit Ehernet) aufbauen. Der in der Platte vorgesehene Prozessor führt eine intelligente Anforderungsverwaltung durch. Der Verwaltungsalgorithmus sollte programmierbar sein; natürlich kann man auch einen nicht programmierbaren Algorithmus verwenden. Da die AD **12** direkt an das Netzwerk **14** angeschlossen ist, kann sie Hackerangriffen ausgesetzt sein. Sie sollte deshalb eigene Zugriffsrechtüberwachungen durchführen.

Zur einfachen Verwirklichung und um doppelten Aufwand zu vermeiden, implementiert das DFS keine eigene Verzeichnisstruktur. Es verwendet die Verzeichnisstruktur des ursprünglichen Dateisystems der Benutzerhosts **16**, beispielsweise NTFS.

Fig. 7 zeigt die Implementierung der DFS Verzeichnisstruktur am Beispiel einer Struktur auf NFTS basierenden Struktur. Natürlich können auch andere Dateisysteme Anwendung finden.

Jedes DFS Volume **46** hat eine LAD **44**, die NTFS versteht. Für jedes "Pseudo-Verzeichnis" im DFS Volume **46** gibt es ein Verzeichnis im NTFS **122**, das auf der entsprechenden LAD **44** des Volumes **46** liegt. Ebenso gibt es für jede DFS Datei eine NTFS Datei im LAD **44**. In dieser Implementierung übernimmt die LAD **44** die Verzeichnisinformation und die Zugriffe darauf und die ADD **42** speichern die Applikationsdaten und geben diese wieder. Benötigt eine Applikation am Benutzerhost eine DFS Datei und/oder ein Verzeichnis (Schritt **110**) leitet der auf dem Benutzer host 116 laufende DFS-Kernel dies an das NTFS weiter (Schritt **112**). Das NTFS kontaktiert die LAD (Schritt **114**) und liest die entsprechende Attributdatei (Schritt **116**), welche dann an das DFS übergeben wird (Schritt **118**). Das DFS reagiert auf die Anforderung dann mit den entsprechenden Daten (Schritt **120**).

Fig. 8 zeigt detailliert einen Lesevorgang im Dateisystem. In Schritt **130** empfängt der Kernel **24** im DFS Klient eine Leseanforderung von einer Anwendung. Ist eine Datei geöffnet, wird die LAD **44** für das Volume um die entsprechende Attributdatei für die angeforderte DFS Datei angefragt. Der die Leseanforderung ausgebende DFS Klient **16** liest die Datei (Schritt **132**) und speichert die Attributdatei in seinem Speicher (Schritt **134**). Die in der Attributdatei gespeicherte Dateiindextabelle wird als Metadatum für die DFS Datei verwendet.

Aus der Leseanforderung berechnet der DFS-Klient mittels der Dateiindextabelle die entsprechende ADD **42** und die LBA auf der ADD **42** (Schritt **163**). Möglicherweise werden für eine Leseanforderung mehrere dieser tuple erzeugt. Der DFS-Klient sendet diese tuple an die entsprechenden ADD **42** (Schritt **148**). Jede ADD **42** liest die entsprechende LBA (Schritt **140**) und sendet sie zum DFS Klienten **16** zurück (Schritt **142**). Der Klient empfängt dann die Daten von den ADD **42**. Nach Lesen (und eventueller Seitenpufferung) der Blöcke sendet der DFS Klient **16** die entsprechende Bytezahl an die Anwendung (Schritt **144**).

Die Schreiboperation wird zweiteilig erläutert. Im ersten Teil wird erläutert, wie eine zuvor nicht bestehende Datei erzeugt wird; im zweiten Teil, wie eine bestehende Datei aktualisiert wird. Der Hauptunterschied zwischen den beiden Teilen liegt darin, daß im ersten Fall noch keine Dateiindextabelle existiert.

Fig. 9 zeigt detailliert den Schreibvorgang. Der DFS-Kernel am Klienten **24** empfängt von einer Anwendung **160** eine Schreibanforderung. Der DFS Klient **16** (bzw. kernel 24) befragt das DFS Volume **46**, auf das die Datei geschrieben werden soll, nach der Verteilungsgruppe und beginnt mit einer zufällig ausgewählten Startplatte. Der Kernel **24** des DFS Klienten liest entweder die i-node-Tabelle vom DFS Volume **46** oder, falls eine solche nicht besteht, erzeugt diese Tabelle (Schritt **162**). Dann wird die i-node-Tabelle in den Speicher des Kernel gespeichert (Schritt **164**). Der erste Block wird auf die Startplatte geschrieben. Nach dem ersten Bock werden die anderen Blöcke reihum auf die Verteilungsgruppe geschrieben. Ein Volume **46** ist dann "voll", wenn eine der beteiligten ADD **42** beim Bedienen einer Schreibanforderung keinen Speicherplatz mehr hat.

Empfängt ein DFS Klient **16** eine Schreibanforderung (Schritt **116**), zerlegt er die angeforderten Bytes in entsprechende DFS Plattenblöcke. Jeder Block wird dann an die ADD **42** in der entsprechenden Folge der Verteilungsgruppe als ein tuple aus drei Feldern gesendet: Die Adresse der ADD **42**, der zu schreibende Datenblock und eine auf (–1) initialisierte LBA (Schritt **166**). Der Grund für diese Initialisierung der LBA auf (–1) liegt darin, daß die ADD **42** angewiesen werden muß, einen freien Block für die Daten freizugeben. Die bevorzugte Ausführungsform verwendet als Initialisierungswert (-1); natürlich können auch andere Vorgehensweisen verwendet werden.

Die ADD **42** prüft, ob sie Platz zum Schreiben des Datenblockes hat. Findet sie einen freien Block, schreibt sie den Block (Schritt **168**). Dann sendet sie die entsprechende LBA als Bestätigung an den DFS Klienten zurück (Schritt **170**). Der DFS Klient **16** verwendet die LBA; um für die geschriebene DFS Datei die Dateiindextabelle aufzubauen. Empfängt der Kernel **24** des DFS Klienten die Bestätigung (Schritt **172**), aktualisiert er die i-Node-Tabelle (Schritt **174**). Um den Schreibvorgang zu beschleunigen, wird die Bestätigung gesendet, bevor der Block tatsächlich auf die Platte geschrieben wurde. Um eine durchgängige Darstellung des Dateisystems zu haben, sollte die Dateiindextabelle idealerweise in die entsprechende Attributdatei zurückgespielt werden, nachdem jeder Block geschrieben wurde (Schritt **176**). Um die Zahl an Zugriffen auf die Dateiindextabelle zu reduzieren, sollte jedoch optimalerweise seltener auf die entsprechende Attributdatei zurückgeschrieben werden (Schritt **176**).

Beim Aktualisieren besteht bereits in der entsprechenden Attpibutdatei die Dateiindextabelle für die Datei. Der DFS Klient **16** liest und speichert die Dateiindextabelle und bildet die entsprechenden Blöcke für die aktualisierte Adresse der ADD **42** und der LBA ab (ähnlich dem Lesevorgang). Zum Aktualisieren sendet der DFS Klient **16** das tuple aus den beim Schreibvorgang erwähnten drei Feldern (wie oben erläutert). Jedoch wird anstelle der (–1) die aktuelle LBA eingesetzt. Dadurch wird der ADD **42** mitgeteilt, die entsprechende LBA auf der Platte zu aktualisieren. Zur Bestätigung wird diese LBA an den DFS Klienten **16** zurückgeleitet.

Die Unterstützung konkurrierender Lesezugriffe bei mehreren DFS Klienten **16** ist einfach, da keine Aktualisierung auf eine etwaige gemeinsame Datenstruktur des Systems nötig ist. Jeder DFS Klient **16** liest und speichert die Attributdatei, die der DFS Datei entspricht, und liest die Daten wie oben erläutert.

Bei DFS werden auch gleichzeitige Schreibzugriffe bei mehreren Klienten **16** unterstützt. Der endgültige Inhalt der Datei hängt jedoch von der Reihenfolge ab, bei der die zusammenhängenden Daten auf die Datei geschrieben werden,

was nicht deterministisch ist. Jeder Schreibzugriff sperrt die gemeinsame Metadatei, insbesondere die Dateiindextabelle, für konkurrierende Schreibzugriffe. Verschiedene Schreibzugriffe können nichtüberschneidende Bereiche der Tabelle separat sperren.

Im Fall konkurrierender Lese- und Schreibzugriffe ist das Hauptproblem, wie der Lesende auf die aktuellste Dateiindextabelle der DFS Datei Zugriff erhalten kann, die der Schreibende erzeugt. Die aktuellste Version der Dateiindextabelle hat immer der auf das DFS Schreibende. Sie wird auf der entsprechenden Attributdatei periodisch eingespielt. Ist der Lesende schneller als der Schreibende, kann er am Dateiende ankommen, bevor der Schreibende die Datei fertiggestellt hat. Da DFS die Geschwindigkeit der Klienten **16** nicht beschränkt, kann nicht sichergestellt werden, daß eine solche Situation nicht auftritt. Die Applikationen sollte sich dessen bewußt sein und etwaige Folgen abfangen; einige Designoptionen können verhindern, daß dies auftritt.

Der Lesende speichert für die DFS Datei die Dateiindextabelle aus der entsprechenden Attributdatei (die auf der LAD liegt). Die Attributdatei enthält eine Markierung, die anzeigt, daß die Datei vom Schreibenden schreibgeschützt wurde. Dies zeigt dem Lesenden, daß er sich darum bemühen muß, die aktuellste Dateiindextabelle während des Vorgangs zu erhalten (indem er wiederholt aus der LAD einliest). Dazu gibt es verschiedene Vorgehensweisen: Der Lesende bittet periodisch nach einer bestimmten Zeitdauer um eine aktualisierte Tabelle. Der Lesende bittet nach einer Zeitdauer, die abhängig von der Bandbreitenbeanspruchung ist, um eine aktualisierte Tabelle. Der Lesende bittet dann um eine aktualisierte Tabelle, wenn er feststellt, daß er nahe dem Ende der aktuellen Tabelle ist. Alternativ kann der Schreibende periodisch an interessierte Lesende eine aktualisierte Tabelle senden.

Der Schreibende erzeugt die Dateiindextabelle, wenn er die Datei in das volume schreibt. Er spielt neue Einträge der Tabelle an die Attributdatei jeweils dann ein, wenn N Blöcke geschrieben sind (N ist eine Konstante). Es sind auch andere Aktualisierungsverfahren denkbar.

Bei DFS sind mehrere Lesende und Schreibende für eine Datei möglich, solange die Bandbreitenanforderungen dies zulassen. Auf Byteniveau kann eine Datei von mehreren Anwendungen gesperrt werden, solange die Bytebereiche sich nicht überschneiden.

Der Zugriffssteuermechanismus steuert die Zahl an Klienten **16**, die für das System zugelassen werden, basierend auf den Resourcenanforderungen des Klienten **16** und der gegenwärtigen Verfügbarkeit dieser Resourcen. Die Bandbreitenverwaltung sorgt sich um das Durchsetzen der vereinbarten Resourcen in Anspruchnahme jedes Klienten **16**. Bei DFS wird die Zugangssteuerung im DFS Klienten **16** und die Bandbreitenüberwachung in den ADD **42** zugeführt.

Als Hauptresourcen werden die Platten- und Netzwerkbandbreiten angesehen. Die Rechenleistung der Systemkomponenten sollte ausreichen, um die verfügbare Platten- und Netzwerkbandbreite jeder Komponente zu unterstützen. Dann können bei voller Auslastung die Platten und Netzwerkschnittstellen voll ausgeschöpft werden. Es sollte ausreichend Speicher für die notwendigen Pufferungen und Speicherungen vorgesehen sein und der Prozessor sollte in der Lage sein, sowohl die Festplatte als auch das Netzwerk beschäftigt zu halten. Dies gilt auch für die ADD **42** und die Klienten **16**. Ein Klient **16** muß in der Lage sein, die geforderten Daten in der geforderten Datenrate zu verarbeiten.

Ein DFS Datenstrom wird hinsichtlich seiner Zugriffskontrolle am DFS-Klienten **16** bearbeitet, wenn die Applikation eine entsprechende DFS-Datei öffnet. Währenddessen kontaktiert der DFS Klient **16** das Volume **46**, um die Verfügbarkeit der Resourcen zu ermitteln. Sind genügend Resourcen vorhanden, ist die Dateiöffnung erfolgreich. Ansonsten wird die Bandbreite nicht zugestanden und die Applikation kann die Datei entweder im Nichtechtzeitmodus öffnen oder später eine Echtzeitmodusöffnung versuchen. Besondere, DFS spezifische Ein/Ausgabekontrollmechanismen für die Geräte ermöglichen es einer Applikation, ihre Bandbreitenzuweisung zu ändern. Der DFS Klient **16** verarbeitet auch diese Anforderungen.

Nachdem einer Anwendung die angeforderte Bandbreite zugeteilt wurde, wird die Zugriffskontrolle jedesmal dann eingeschaltet, wenn die Anwendung eine Ein/Ausgabe-Operation mit dieser Datei durchführt (Lesen oder Schreiben). Wenn die Datei geschlossen wird, werden die zugewiesene Resourcen der Anwendung entzogen die Datei geschlossen wird. Die Zugriffskontrolle wird bei einer DFS-Datei im Falle folgender Ereignisse durchgeführt. Wenn eine Datei geöffnet wird, wenn eine Bandbreitenanforderung abgesetzt wird, wenn auf eine Datei zugegriffen wird, und wenn eine Datei geschlossen wurde.

Betriebssysteme unterstützen im allgemeinen keine Verfahren zum Ausweiten der Öffnungs-, Lese- und Schreibanforderungen. Deshalb sollte jeder Datei eine Standarddatenlieferungsbandbreite als Attribut entweder der Datei, ihres Verzeichnisses oder ihres Volumes, auf dem sie liegt, zugewiesen werden. Weiter sollte ein Verfahren vorhanden sein, um die reservierte Bandbreite während der Bearbeitung einer offenen Datei zu ändern.

Die Datenlieferungsbandbreite wird als minimale mittlere Datenrate definiert, die am Speicher des Anfordernden ankommen muß, damit die Daten für ihn brauchbar sind. Es ist davon auszugehen, daß diese Rate durch die Netzwerkschnittstellenhardware und die Netzwerksoftware des Anfordernden begrenzt ist.

Mit diesen Rahmenbedingungen wird AC durchgeführt, wenn eine Datei geöffnet oder geschlossen ist, oder wenn die Bandbreitenanforderung explizit bei der Bearbeitung einer Datei geändert wird. Wird eine Datei geöffnet, wird AC ausgehend von der Standarddatenrate der Datei durchgeführt. Ist keine ausreichende Bandbreite verfügbar, wird die Datei ebenfalls geöffnet, jedoch ohne Bandbreitenreservierung (AC weist die Anforderung zurück). Etwaige Dateizugriffe werden gesperrt, bis die Datenrate verfügbar ist. Dadurch kann die Applikation eine andere Bandbreite anfordern, indem eine Anforderung zur Zuweisungsänderung vom aktuellen oder dem Standardwert auf einen anderen Wert abgesetzt wird. AC wird dann eingesetzt, um die geforderte Bandbreite zu reservieren. Ist dies erfolglos, wird die Dateibearbeitung ebenfalls als ohne Bandbreite erfolgend markiert. Zugriffe auf eine Datei ohne reservierte Bandbreite werden gesperrt, bis die Bandbreite verfügbar ist. Die Anwendung wird gesperrt, wenn sie den ersten Zugriff (Lesen oder Schreiben) absetzt. Eine Anwendung kann die Bandbreitenzuweisung explizit auf Null setzen, so daß Anfragen mit der bestmöglichen Bandbreite abgearbeitet werden. AC ist zum Freigeben der Bandbreitenreservierung auch dann involviert, wenn die Datei geschlossen wird.

Die Standarddatenlieferungsbandbreite für eine neue Datei wird mittels eines einfachen Ableitungsmechanismus ermittelt. Das Stammverzeichnis hat eine Standarddatenlieferungsbandbreite, die als eines der Attribute bei der Initialisierung des DSF Volumes **46** definiert wurde. Jede neue Datei und jedes neue Verzeichnis leitet die Standardbandbreite von seinem Stammverzeichnis ab. Der Standardwert kann mit betriebssystemabhängigen oder applikationsprogrammabhängigen Befehlen geändert werden.

Die Zugriffskontrolle entscheidet nur, ob eine Anforde-

rung die Lieferrandbedingungen der Datei, die bereits zugelassen wurden, nicht verletzt, und ob die Anforderungen für die neue Datei ebenfalls sichergestellt werden können. Die Zugriffskontrolle verwaltet die Bandbreiteninanspruchnahme einer besonderen Anwendung, die auf eine Datei wirkt, nicht. Deshalb ist eine Bandbreitenverwaltung notwendig. Der Hauptzweck dieser Verwaltung liegt darin, sicherzustellen, daß eine Anwendung nicht mehr Resourcen (Platten- oder Netzwerkbandbreite) bekommt, als sie anfordert.

Die Bandbreitenbegrenzung kann entweder im Klient **16** oder in den ADD **42** durchgeführt werden. Für beste Wirksamkeit sollte sie in den ADD **42** erfolgen. Das DFS unterstützt einen "pull" mode der Anforderungen. In diesem Modell kann eine Anwendung Daten mit einer Rate anfordern, die höher als die vereinbarte ist. Die Anforderungen der Anwendungen sollten jedoch nur mit der vereinbarten Bandbreite erfüllt werden. Eine Anwendung kann eine Spitzenbandbreite in Anspruch nehmen, die höher als die vereinbarte ist. Auf lange Sicht gesehen, stellt die Bandbreitenverwaltung jedoch sicher, daß die mittlere Inanspruchnahme nahe dem vereinbarten Wert liegt. Dies wird erzwungen, wenn jeder Blockanforderung eine Frist zugewiesen wird. Die Frist wird aus der zugestandenen Bandbreite berechnet. Das DFS stellt sicher, daß die Anforderung nicht vor der Frist abgearbeitet wird, wenn das System geladen ist. Die Frist ist nicht absolut, da das Betriebssystem des host keine Echtzeitleistungen garantieren kann.

DFS leistet Fehlertoleranz auf verschiedenen Komponentenniveaus. Fehlertoleranz auf Speicher- und Netzwerkniveau sind sehr wichtig für die Art von Anwendungen, die mit DFS unterstützt werden sollen.

Bei DFS kann eine softwarebasierte Redundanz im Speicher oder in der Hardware (RAID) oder beides verwendet werden. Ein DFS Volume **46** kann von einer der folgenden Arten sein: Nichtfehlertolerant, softwaregespiegelt oder Hardware-RAID. Ein nichtfehlertolerantes Volume ist die Standardkonfiguration. Daten können im Falle eines Plattenversagens im Volume nicht wiederhergestellt werden. Ein softwaregespiegeltes Volume hat für jede Platte des Volumes im System einen identischen Spiegel. Dies ist eine Softwareemulation des RAID Levels **1**, jedoch ohne besondere RAID-Hardware. Eine Verwirklichung des nötigen Schemas erfordert eine geringe Modifikation der DFS-Datenstrukturen. Ein DFS Volume kann an das Netzwerk auch als echte RAID-Platte angeschlossen werden. Die Hardware kann jedes beliebige RAID Level aufweisen und muß DFS nicht bekannt sein.

Eine fehlertolerante Datenverbindung (fehlertolerante Ethernet Netzwerk-Karten und Treiberprogramme) gewährleistet die Sicherheit in DFS auf Netzwerkniveau. Diese Schicht, die in der Hierarchie tiefer liegt, modifiziert keine Funktionalität des DFS.

DFS kennt zwei Prioritätsniveaus: Echtzeit und Nichtechtzeit. Der Unterschied in diesen beiden Niveaus liegt darin, daß Laufzeit und Durchsatz nur für die Echtzeitanwendungen garantiert werden. DFS verwendet eine einfache Notation, um die Priorität einer Anwendung festzustellen. Hat eine Anwendung Null Bandbreite für ihren Betrieb zugewiesen, wird sie als Echtzeitanwendung angesehen, ansonsten als Nichtechtzeitanwendung. Mittels gerätespezifischer DFS Ein/Ausgabensteuerungsanforderungen kann eine Anwendung dynamisch zwischen zwei Klassen umschalten. Die Priorität wird nur bei Festplattenzugriffen und Netzwerk (queues) sichergestellt. Da keine Unterstützung vom eigentlichen Betriebssystem gegeben ist, kann keine Priorisierung auf Prozeßausführungsebene garantiert werden.

Jedes Dateisystem sollte zum Schutz der Benutzer einen gewissen Grad an Zugriffsicherheit haben. Sicherheit ist vor allem deshalb bei DFS wichtig, da die ADD **12** direkt an das Netzwerk **14** angeschlossen ist, und damit Hackerangriffen stärker ausgesetzt sind. DFS hat mehrere Sicherheitsebenen.

DFS verwendet das ursprüngliche Dateisystem zum Verzeichnis und Dateimanagement (das auf der LAD mit der Attributdatei aufbaut). Alle Datei- und Verzeichnisrechte und deren Sicherheitsmechanismen sind deshalb bei DFS verwendbar.

Bei DFS sind die ADD **42** direkt an das Netzwerk **14** angebunden. Obwohl ein vollständiges Dateisystem auf einer bestimmten ADD **42** vorhanden ist, kann ein unauthorisierter Benutzer **16** Blocks von den ADD **42** lesen. Damit sind diese anfälliger auf Hackerangriffe als konventionelle Systeme. Das besondere Klientenauthorisierungsverfahren stellt die Identität des Klienten **16** sicher, der auf eine ADD **42** zugreift.

Datensicherheit wird durch eine geeignete Verschlüsselung der Daten gewährleistet. Obwohl die Konstruktion von DFS solche Verschlüsselung und Entschlüsselung unterstützt, wird sie normalerweise in DFS aufgrund des großen Verwaltungsaufwandes bei der Verschlüsselung und Entschlüsselung der Daten nicht verwirklicht.

Patentansprüche

1. Verteiltes Dateisystem zum Informationszugriff von einem host-System über ein Netzwerk (**14**) auf ein Speichersystem (**10**) mit:
   - einem im Speichersystem (**10**) liegenden Speichersystemagenten (**20**, **22**, **24**, **26**) mit einem freien Listenmanagementsystem, das die physikalische Speicherstelle der im Speichersystem (**10**) gespeicherten Informationen feststellt,
   - einem im host-System (**16**) liegenden Verzeichnisstruktursystem, das eine logische Organisation mehrerer Dateien festlegt, die den am Speichersystem (**10**) abgelegten Informationen entsprechen,
   - einem mit dem Netzwerk (**14**) verbundenen Legacyattributdatenspeicher (**44**), der den im Speichersystem (**10**) gespeicherten Informationen zugeordnete Metadaten speichert und aus dem die physikalische Speicherstelle der gespeicherten Informationen festgestellt werden kann, und
   - einem im host-System (**16**) liegenden Klientenagenten (**24**), der Zugriff auf die Metadaten des Legacyattributdatenspeichers (**44**) hat und mit dem Verzeichnisstruktursystem zusammenwirkt, um Dateien den entsprechenden physikalischen Speicherstellen zuzuordnen, wodurch auf die den Dateien entsprechenden Informationen am Speichersystem (**10**) zugegriffen und diese an das host-System (**16**) geliefert werden.
2. Verteiltes Dateisystem nach Anspruch 1, dadurch gekennzeichnet, daß das Speichersystem (**10**) mindestens eine autonome Platte (**12**) mit einem zugeordneten Prozessor (**25**), der den Speichersystemagenten (**26**) verwirklicht, aufweist.
3. Verteiltes Dateisystem nach Anspruch 1, dadurch gekennzeichnet, daß das Speichersystem ein serverloses Speichersystem ist, das mindestens eine autonome Platte (**12**) mit einem zugeordneten Prozessor (**25**), der den Speichersystemagenten (**26**) verwirklicht, aufweist.
4. Verteiltes Dateisystem nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, daß der Speichersystemagent (**20**) weiter ein Netzwerkprotokollsystem

aufweist, mittels dem das Speichersystem eine Kommunikationsverbindung mit dem Netzwerk (**14**) hat.

5. Verteiltes Dateisystem nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, daß der Speichersystemagent (**20**) weiter ein Zugriffsrechtüberwachungssystem (**22**) hat, das Zugriffe aus dem Netzwerk (**14**) auf das Speichersystem (**10**) überwacht.

6. Verteiltes Dateisystem nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, daß der Speichersystemagent (**20**) weiter ein Anforderungssteuersystem hat, das die Reihenfolge, mit der Informationszugriffsanforderungen abgearbeitet werden, vorgibt.

7. Verteiltes Dateisystem nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, daß der Klientenagent (**16**) weiter ein Zugriffssteuerungssystem hat, das Zugriffe auf die Dateien steuert und einen Liefermodus festlegt, mit dem eine angeforderte Datei geliefert werden kann.

8. Verteiltes Dateisystem nach Anspruch 7, dadurch gekennzeichnet, daß der Liefermodus ein Echtzeitmodus ist.

9. Verteiltes Dateisystem nach Anspruch 7 oder 8, dadurch gekennzeichnet, daß das Zugriffssteuersystem den Zugriff auf die Dateien ausgehend von Netzwerkbandbreitenauslastung und Plattenbandbreitenauslastung steuert.

10. Verteiltes Dateisystem nach einem der vorherigen Ansprüche, dadurch gekennzeichnet, daß das Speichersystem (**10**) eine die Daten speichernde Einheit (**46**) umfaßt, welche eine autonome Platte (**42**), die Applikationsdaten, und eine Legacyplatte (**44**), die Metadaten speichert, umfaßt.

11. Verteiltes Dateisystem nach Anspruch 10, bei dem die Applikationsdaten über mehrere autonome Datenplatten (**42**) verteilt werden.

12. Verteiltes Dateisystem nach Anspruch 11, dadurch gekennzeichnet, daß die Legacyplatte (**44**) Metadaten für mindestens eine Einheit (**46**) speichert.

13. Verfahren zur Informationsübermittlung über ein Netzwerk zwischen einem Speichersystem und einem host-System mit folgenden Schritten:

- Bereitstellen eines im Speichersystem liegenden Speichersystemagenten mit einem freien Listenmanagementsystem, das die physikalische Speicherstelle der im Speichersystem gespeicherten Informationen feststellt,
- Bereitstellen eines im host-System liegenden Verzeichnisstruktursystems, das eine logische Organisation mehrerer Dateien festlegt, die den am Speichersystem abgelegten Informationen entsprechen,
- Bereitstellen eines mit dem Netzwerk verbundenen Legacyattributdatenspeicher, der den im Speichersystem gespeicherten Informationen zugeordnete Metadaten speichert und aus dem die physikalische Speicherstelle der gespeicherten Informationen festgestellt werden kann und
- Bereitstellen eines im host-System liegenden Klientenagenten, der Zugriff auf die Metadaten des Legacyattributdatenspeichers hat und mit dem Verzeichnisstruktursystem zusammenwirkt, um Dateien den entsprechenden physikalischen Speicherstellen zuzuordnen, wodurch auf die den Dateien entsprechenden Informationen am Speichersystem zugegriffen und diese an das host-System geliefert werden.

14. Verfahren nach Anspruch 13, gekennzeichnet durch folgende Schritte:

Senden einer ersten Leseanforderung für eine Datei an den Klientenagenten, erstmaliges Anfragen des Legacyattributdatenspeichers für der Datei zugeordnete Metadaten,

Übersetzen der ersten Leseanforderung in mindestens eine erste Sendeanforderung für einen Datenblock basierend aus den beim erstmaligen Anfragen erhaltenen, zugeordneten Metadaten,

Senden der ersten Sendeanforderung an das Speichersystem und Empfangen eines Datenblocks vom Speichersystem.

15. Verfahren nach Anspruch 14, dadurch gekennzeichnet, daß das Speichersystem weiter eine Einheit aufweist, die mindestens eine verteilte Datenplatte zum Speichern von Applikationsdaten und eine Legacyattributdatenplatte zum Speichern von Metadaten aufweist, und der Schritt des Übersetzens das Übersetzen der ersten Leseanforderung in mindestens eine erste Sendeanforderung für mindestens eine verteilte Datenplatte umfaßt.

16. Verfahren nach Anspruch 14 oder 15, gekennzeichnet durch folgende Schritte:

Senden einer zweiten Leseanforderung für eine Datei an einen weiteren Klientenagenten,

zweitmaliges Anfragen des Legacyattributdatenspeichers für der Datei zugeordnete Metadaten,

Übersetzen der zweiten Leseanforderung in mindestens eine zweite Sendeanforderung für einen Datenblock basierend aus den beim zweiten Anfragen erhaltenen zugeordneten Metadaten,

Senden der zweiten Sendeanforderung an das Speichersystem und Empfangen eines Datenblocks vom Speichersystem.

17. Verfahren nach Anspruch 13, gekennzeichnet durch folgende Schritte:

Senden einer ersten Schreibanforderung an den Speichersystemagenten,

Senden einer freien Blockadresse an das host-System auf diese erste Schreibanforderung hin,

Schreiben einer Datei in das Speichersystem,

Erzeugen einer der Datei zugeordneten Dateiindextabelle und

Senden der Dateiindextabelle an den Legacyattributdatenspeicher.

18. Verfahren nach Anspruch 17, bei dem das Senden der Dateiindextabelle deren abschnittsweises Senden während der Erzeugung der Dateiindextabelle umfaßt.

19. Verfahren nach Anspruch 18 gekennzeichnet durch folgende Schritte:

Setzen einer Markierung im Legacyattributdatenspeicher zum Anzeigen, daß in die Datei geschrieben wird,

Senden einer konkurrierenden Leseanforderung an den Speichersystemagenten, um die Datei zu lesen,

Senden der Dateiindextabelle und der Markierung an das host-System auf die Leseanforderung hin,

Lesen der Datei und

Anfordern von Aktualisierungen der Dateiindextabelle während des Lesens der Datei.

20. Verfahren nach Anspruch 13, gekennzeichnet durch folgende Schritte:

Senden einer konkurrierenden Schreibanforderung an den Speichersystemagenten,

Senden der Dateiindexiabelle der Datei auf die konkurrierende Schreibanforderung hin, wobei die Dateiindextabelle mindestens zwei Teilen der Datei entsprechende Abschnitte hat,

Sperren diejeniger Abschnitte der Dateiindextabelle, die dem zu schreibenden Teil der Datei entsprechen,

Schreiben in diesen Dateiteil,
Aktualisieren der der Datei zugeordneter Dateiindextabelle und
Senden der Dateiindextabelle an den Legacyattributdatenspeicher.

21. Verfahren nach Anspruch 13, gekennzeichnet durch folgende Schritte:
Senden einer von einer Anwendung stammenden Anforderung für Verzeichnisinformation an den Klientenagenten,
Senden der Anforderung an das Verzeichnisstruktursystem,
Abfragen des Legacyattributdatenspeichers, um eine Attributdatei, die der angeforderten Verzeichnisinformation entspricht, zu lesen,
Empfangen der Attributdatei, Senden der Attributdatei an den Klientenagenten und
Liefern der geforderten Verzeichnisinformation an die Anwendung.

Hierzu 6 Seite(n) Zeichnungen

- Leerseite -

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

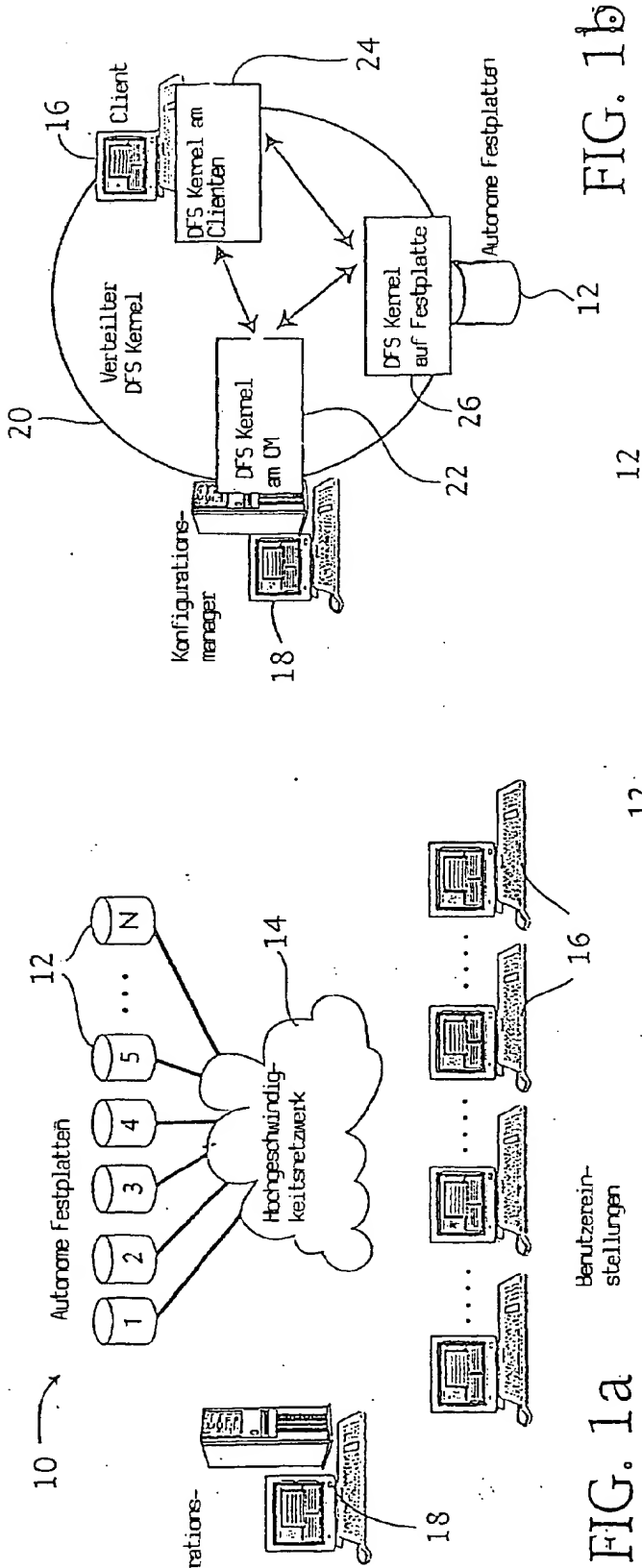


FIG. 1a

FIG. 1b

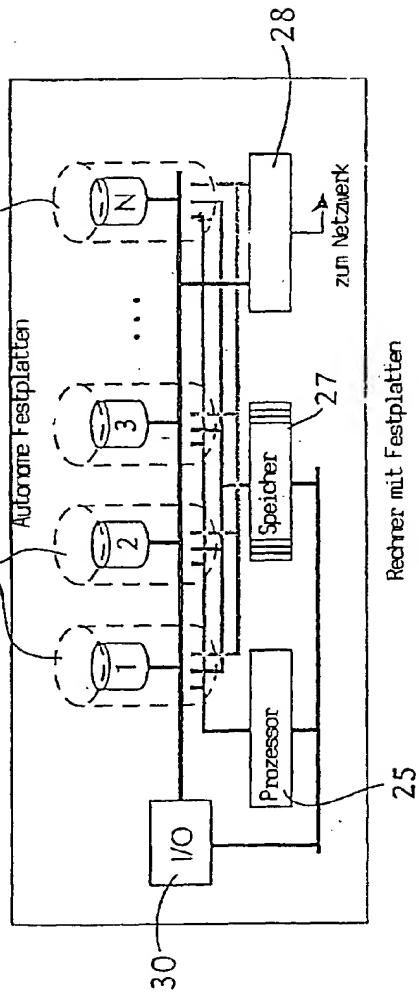


FIG. 1c

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

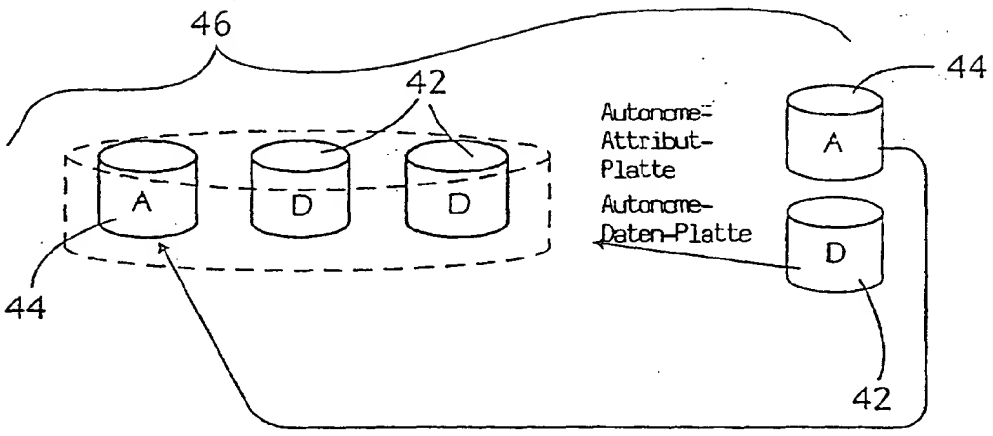


FIG. 2

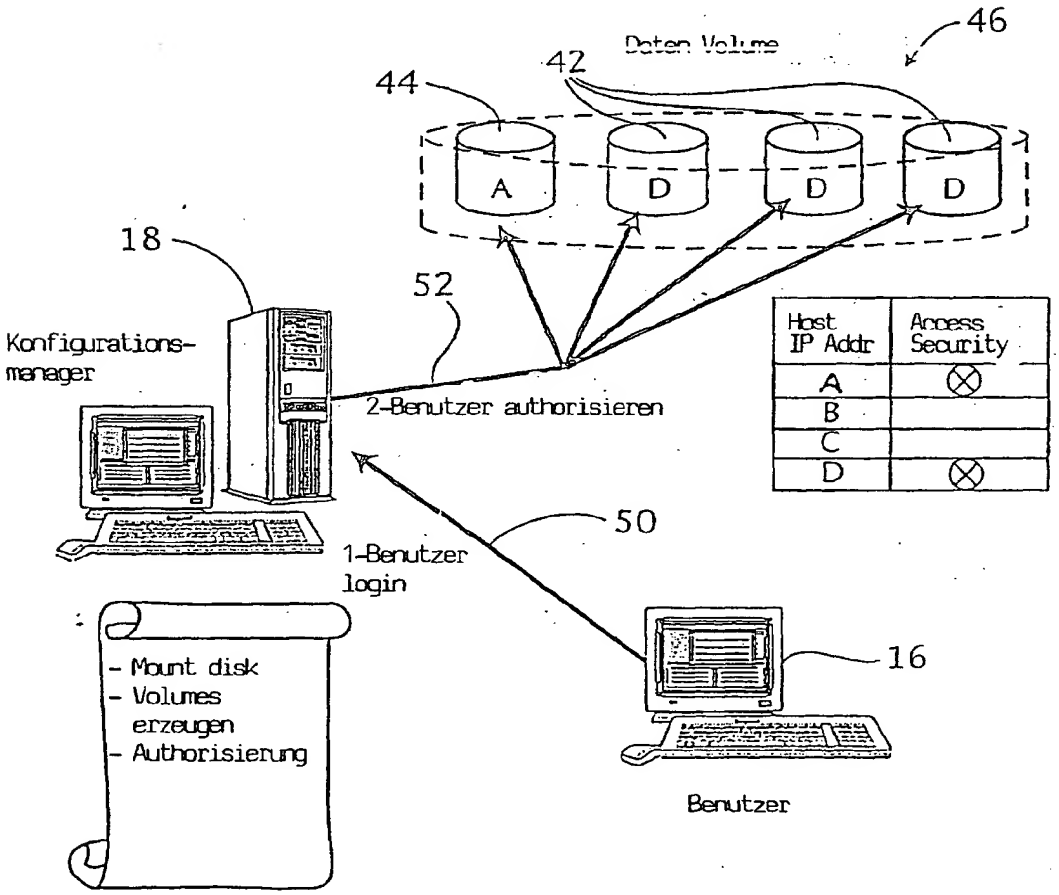


FIG. 3

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

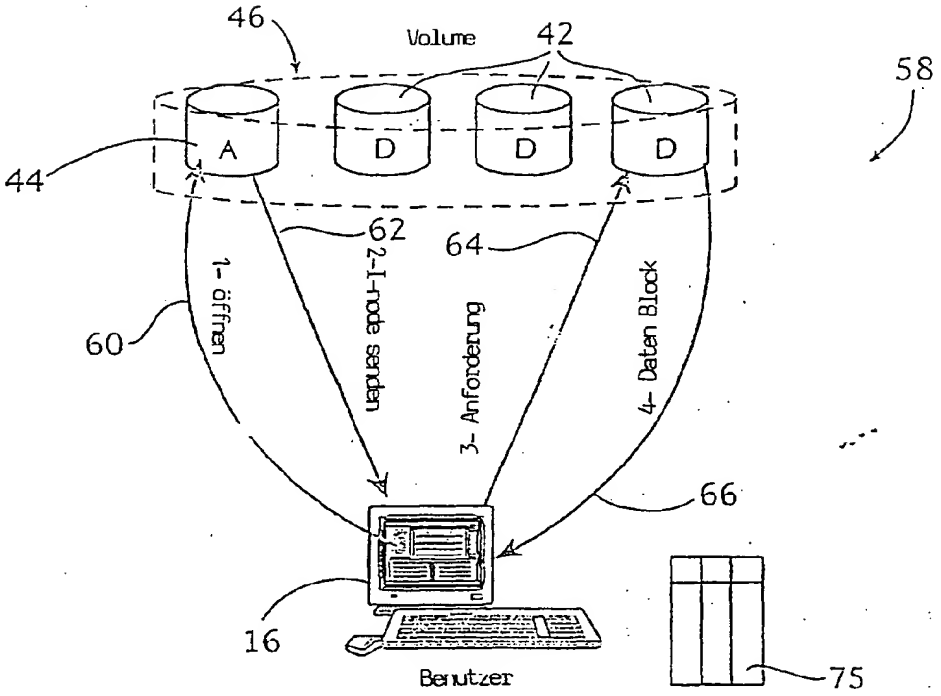


FIG. 4

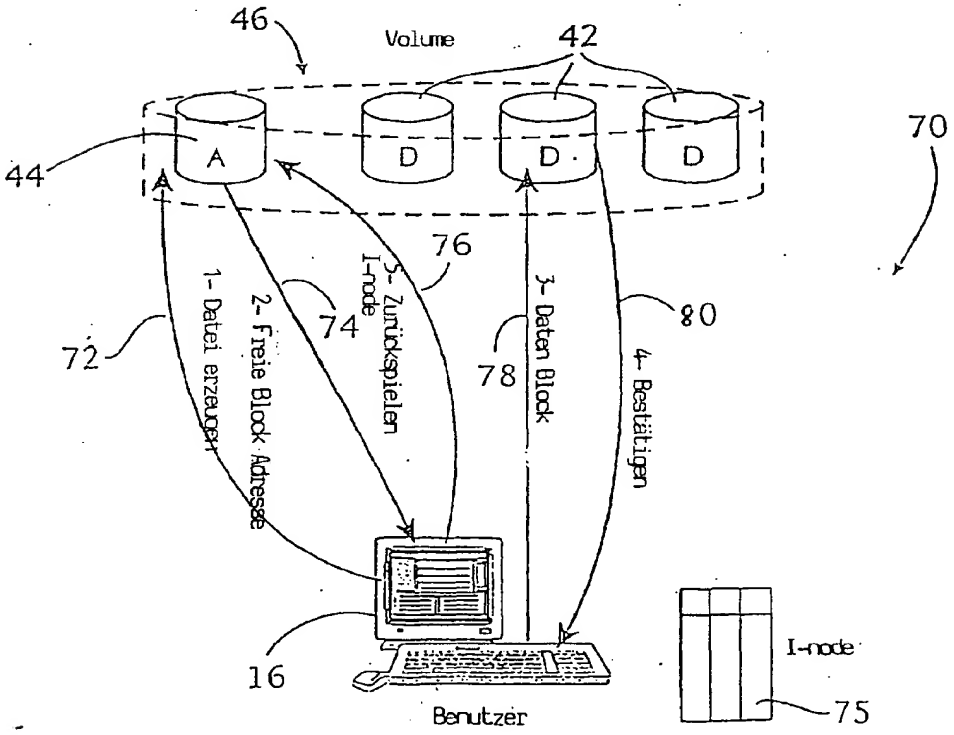


FIG. 5

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

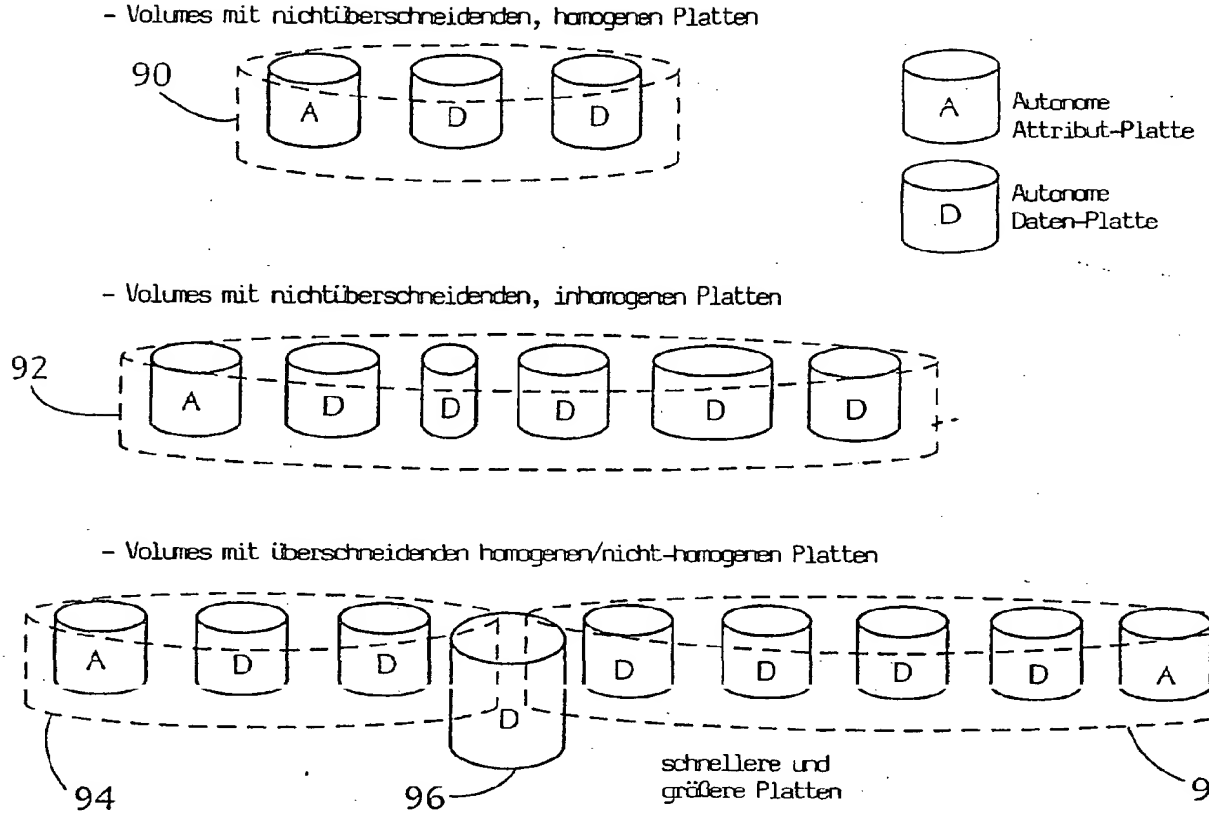


FIG. 6

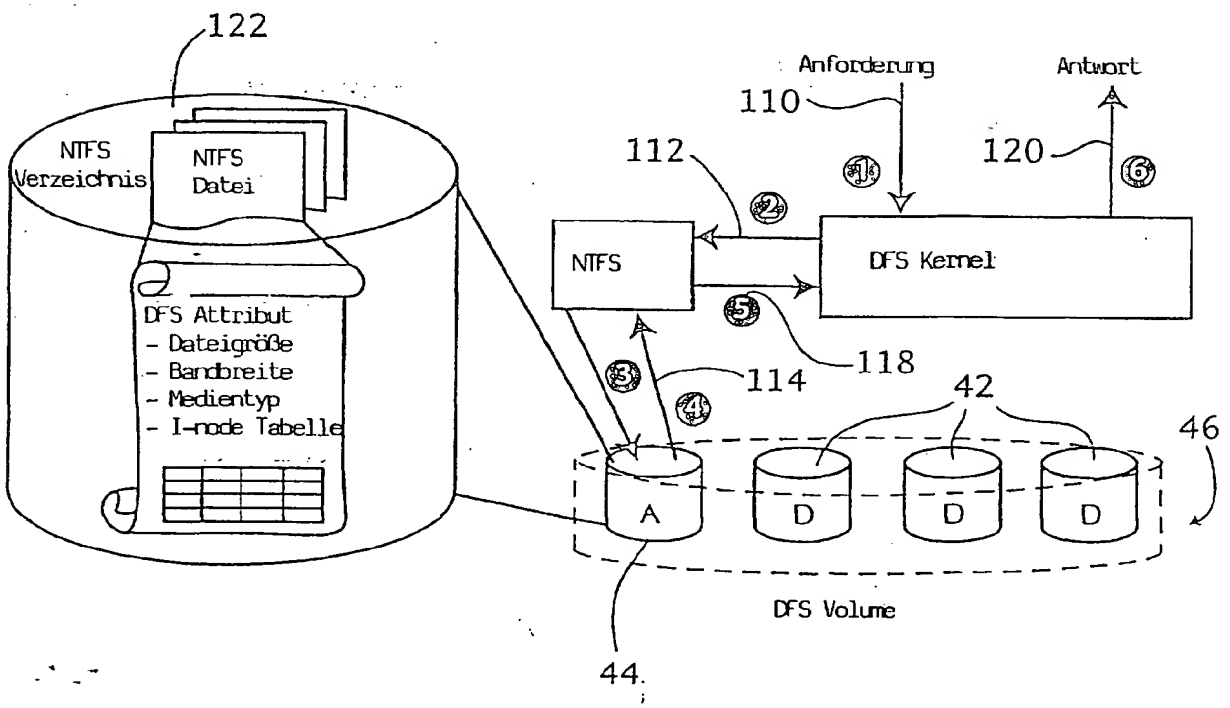


FIG. 7

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

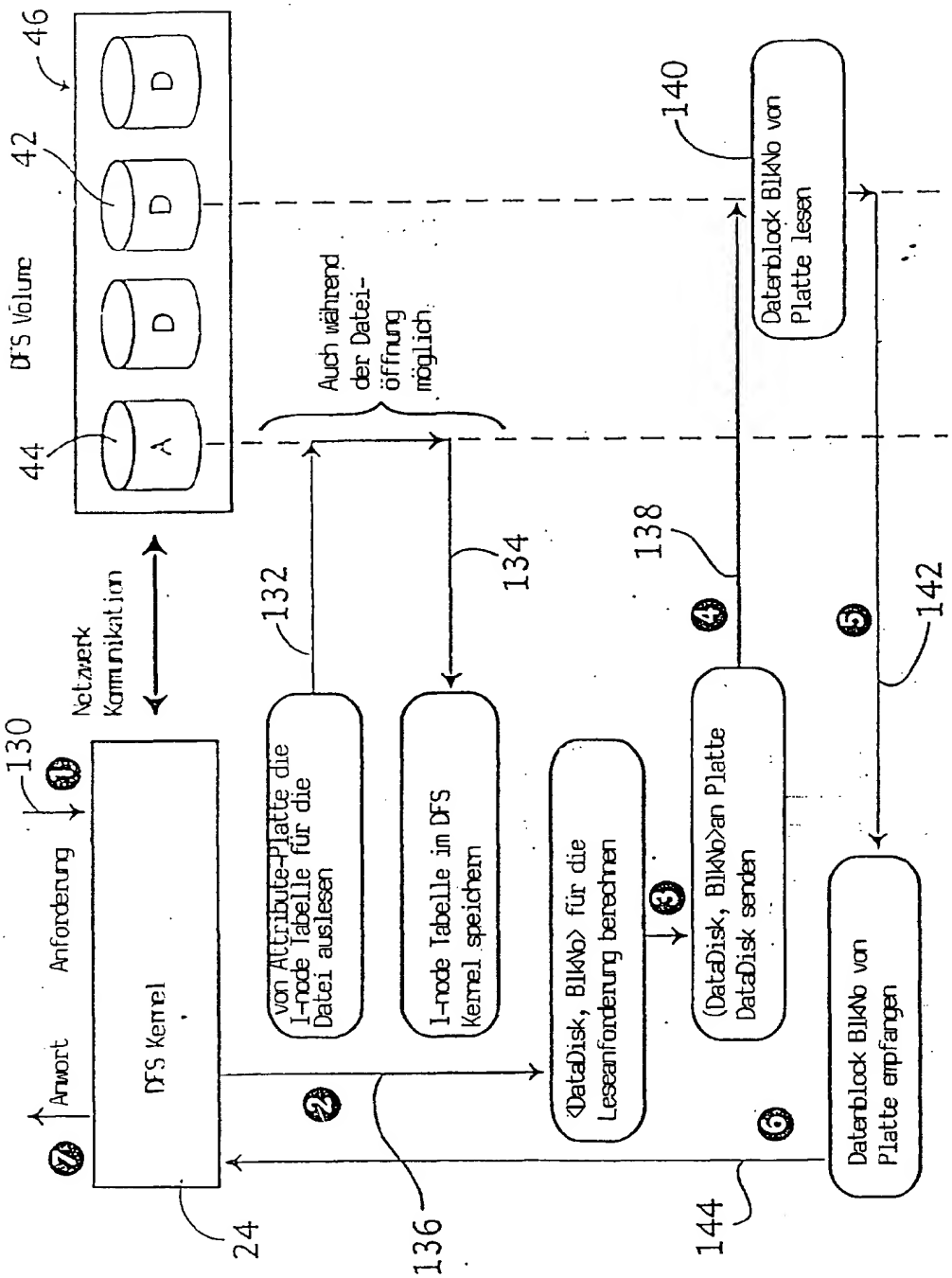


FIG. 8

Nummer: DE 100 32 962 A1
Int. Cl.[7]: G 06 F 17/30
Offenlegungstag: 15. März 2001

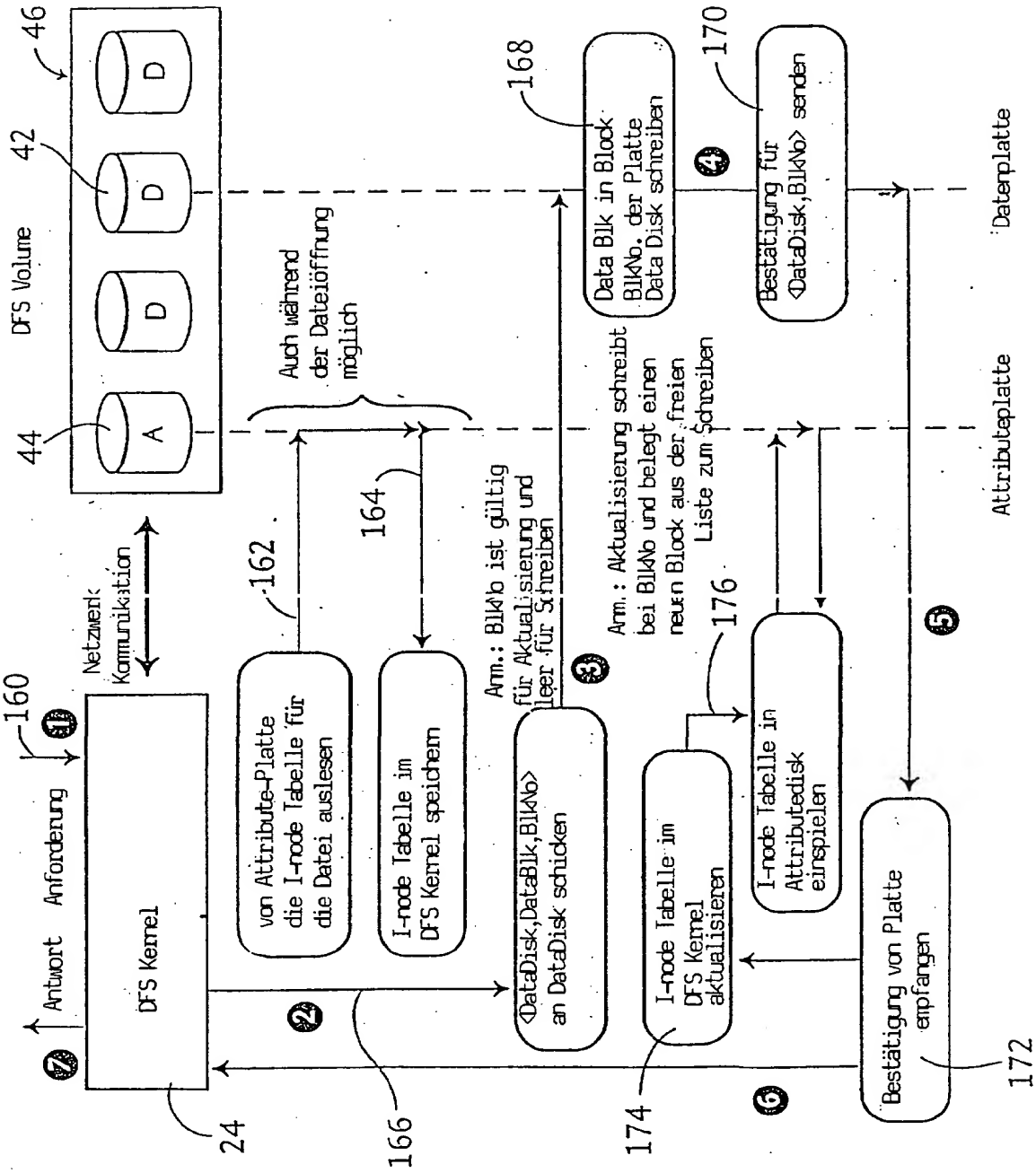


FIG. 9